Time Triggered Offline Scheduling of Data Dependent Real-Time Tasks Accounting for the Preemption and Scheduler Cost

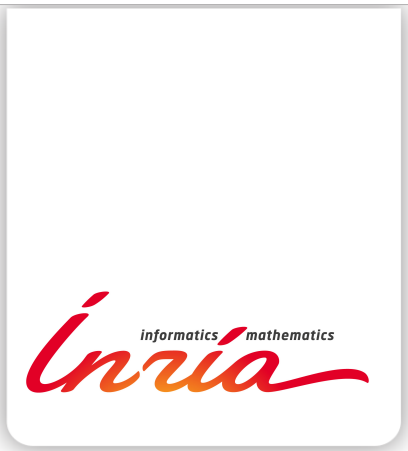
Yves Sorel, Falou Ndoye, Walid Talaboulma, Mehdi Mezouak, Liliana Cucu-Grosjean

Time Triggered Offline Scheduling of Data Dependent Real-Time Tasks Accounting for the Preemption and Scheduler Cost

Yves Sorel, Falou Ndoye, Walid Talaboulma, Mehdi Mezouak, Liliana Cucu-Grosjean

RESEARCH REPORT
N° 9318
December 2019
Project-Team Kopernic

ISSN 0249-6399 ISRN INRIA/RR--9318--FR+ENG



Time Triggered Offline Scheduling of Data Dependent Real-Time Tasks Accounting for the Preemption and Scheduler Cost

Yves Sorel, Falou Ndoye, Walid Talaboulma, Mehdi Mezouak, Liliana Cucu-Grosjean

Project-Team Kopernic

Research Report n° 9318 — December 2019 — 26 pages

Abstract: Time critical embedded systems usually consist of a set of periodic data dependent real-time tasks issued from a functional specification achieved with a block diagram language. Although non-preemptive real-time scheduling is safer than preemptive real-time scheduling in a time critical context, preemptive real-time scheduling may provide better scheduling ratio. However, this better scheduling ratio comes with a cost related to the preemption, and a non precise determination of this cost can prevent to satisfy real-time constraints. In this paper we propose an offline schedulability analysis for a set of data dependent periodic tasks which precisely accounts for the preemption and scheduler cost. This analysis produces a scheduling table that is exploited by a time triggered offline scheduler. We show that this scheduler, implemented on an ARM Cortex-M4 bare metal uniprocessor for which the preemption and scheduler cost is precisely determined, is able to schedule correctly a set of tasks that can miss some deadlines if the preemption and scheduler cost is incorrectly determined. Therefore, such offline scheduling and scheduler approach is perfectly suited for time critical embedded systems.

Key-words: critical embedded system, real-time scheduling, data dependent tasks, preemption cost, scheduler cost, offline real-time schedulability analysis, time triggered scheduler, ARM Cortex-M4 processor

RESEARCH CENTRE
PARIS

2 rue Simone Iff - CS 42112
75589 Paris Cedex 12

Ordonnancement temps réel hors ligne de tâches avec dépendances de données prenant en compte le coût de la préemption et de l'ordonnanceur

Résumé : Les systèmes temps réel critiques consistent en général en un ensemble de tâches périodiques avec des dépendances de données issu d'une spécification fonctionnelle réalisée à l'aide d'un langage de type schéma bloc. Bien que l'ordonnancement temps réel non préemptif soit plus sûr que l'ordonnancement temps réel préemptif dans un contexte critique, l'ordonnancement temps réel préemptif peut avoir de meilleurs taux d'ordonnancement. Cependant, cela a pour conséquence un coût associé à la préemption. Ainsi, la non prise en compte précise de ce coût peut empêcher de satisfaire les contraintes temps réel. Dans ce papier nous proposons une analyse d'ordonnançabilité pour un ensemble de tâches avec des dépendances de données qui prend en compte précisément le coût de la préemption et de l'ordonnanceur. Cette approche produit une table d'ordonnancement qui est exploitée par un ordonnanceur hors ligne déclanché par le temps. Nous montrons que cet ordonnanceur, implanté sur un monoprocesseur nu ARM Cortex-M4 pour lequel le coût de la préemption et de l'ordonnanceur est précisément déterminé, est capable d'ordonnancer correctement un ensemble de tâches dont certaines peuvent rater leur échéance si ce coût n'est pas correctement déterminé. Ainsi, cette approche hors ligne d'ordonnancement et d'ordonnanceur est parfaitement adaptée pour les systèmes temps réel critiques.

Mots-clés : système embarqué critique, ordonnancement temps réel, tâches avec dépendances de données, coût de la préemption, coût de l'ordonnanceur, analyse d'ordonnançabilité temps réel hors ligne, ordonnanceur déclanché par le temps, processeur ARM Cortex-M4

1 Introduction

In this paper, we study time critical embedded systems, i.e. systems for which time constraints should be satisfied in order to avoid catastrophic consequences. Such systems may be defined by periodic tasks with data dependances constraints. These tasks are programs that are manually or automatically obtained from a functionnal specification, usually achieved with tools such as Simulink [2], Scade [1], or other modelling tools based on block diagrams. Indeed, the functional specification describes the functions that must be executed by the system as well as the dependences carrying the data produced and consumed by these functions. Such dependences may involve a precedence relation on the execution of every producer function relatively to one or several consumer functions, and lead to sharing the transfered data. Hence, every data dependent function is associated to a program that is temporally characterised to become a data dependent real-time task. Some of these characteristics like release times, periods and deadlines do not depend on the processor where they will be executed, whereas all WCET (Worst Case Execution Time) of the tasks depend on the processor where they will be executed. It is worth noting that the processors considered in this paper have only one core. Similarly, worst case communication times (WCCT) depend on the communication link where a data dependence will be executed. Usual schedulability analyses of periodic data dependent tasks are based on the WCET, and thus, their estimations require that the designer determines accurately the WCET which is strongly related to the internal architecture of the processor. The more this architecture is complex the more it is difficult to determine the WCET.

Although non-preemptive real-time scheduling is safer than preemptive real-time scheduling in a time critical context, preemptive real-time scheduling provides a better scheduling ratio. However, the preemption has a cost that is often neglected or roughly approximated. Similarly, the scheduler has a cost which is neglected or roughly approximated.

Whatever the method to estimate WCETs and WCCTs, taking into account the preemption and scheduler cost amounts generally to add a certain value to their estimated values. Common industrial usage consists in adding a high-water margin to the largest observed value of the execution time of a task as a percentage of this observed value, hoping that such margin covers the preemption and scheduler cost. However, the high-water approach has no justification and it may lead to missed deadlines during the execution of the task even though schedulability conditions have been satisfied. In the best case where deadlines are not missed resources could be wasted when this percentage is chosen to high.

Our contribution We propose a schedulability analysis of periodic data dependent tasks scheduled on a uniprocessor and give the corresponding algorithm. Our analysis takes into account the precise cost of the preemption by counting the number of preemptions within a schedulability interval, as well as the cost of the scheduler. This solution guarantees the real-time schedulability of a set of periodic data dependent tasks while minimizing the needed resources, along the schedulability interval. Our schedulability analysis produces, as well, a scheduling table that is exploited by a time triggered offline scheduler. We provide the principles of this scheduler and the corresponding interruption routine which is implemented on an ARM Cortex-M4 bare metal uniprocessor. Finally, we present a performance evaluation of our approach that shows on the one hand that the time triggered scheduler running on this processor exploits correctly the scheduling table produced by the offline schedulability analysis, i.e. schedules the tasks as predicted in the scheduling table, while taking into account the preemption and scheduler cost. On the other hand, the performance evaluation shows that this time triggered scheduler provides a correct schedule for a set of tasks that cannot be correctly scheduled when the preemption and scheduler cost is neglected.

Organization of the paper The remainder of the paper is organized as follows. In Section 2 we present the related work with respect to periodic data dependent tasks and to the preemption and scheduler cost. We introduce the model and the associated notations in Section 3. In Section 4 we present the proposed offline schedulability analysis and the corresponding algorithm. In Section 5 we provide the principles of the time triggered offline scheduler which exploits the scheduling table produced by the schedulability analysis, and the corresponding algorithm. We give in annexe the flowchart and the C and assembly program of the corresponding interruption routine. In Section 6 we provide a performance evaluation of the proposed scheduler on an ARM Cortex-M 4 bare metal processor. Finally, Section 7 contains our conclusions and some directions for future work.

2 Related Work

We assume that the processor where the real-time tasks will be executed, has neither cache nor complex pipeline or specific internal architecture features. The previous assumptions are usually made in time critical embedded systems where determinism is a key issue. A real-time embedded system is said deterministic when it executes on a processor, if it will always produce the same output values when it consumes the same input values. For such processor, the preemption cost corresponds to the duration necessary to save the context of the preempted task and the duration necessary to restore this context when the preempted task will be selected again to resume its execution. The scheduler cost depends on its type, online or offline. Due to its cost, a preemption and a scheduler call increase the response time of the preempted task that may cause another preemption, and so on. The preemption and scheduler cost is usually approximated in the WCET as assumed, explicitly, by Liu and Layland in their pioneering article [18]. That is, some percentage of the WCET, is added to its actual value. Roughly speaking, the WCET of a task corresponds to the longest path in the sequential program associated to this task when using static methods or to its largest measured value when using measurement based methods, or a combination of both methods when using hybrid methods. For a survey on these methods see [23]. When the preemption and scheduler cost is neglected, meaning this percentage is close to zero, although a set of tasks verifies the schedulability condition associated to the scheduling algorithm - which will be implemented in the real-time scheduler -, some deadline misses may occur. In order to tackle this problem, a first solution consists in determining the maximum number of preemptions as proposed in [4] or in determining the number of preemptions but without accounting for the cost of each preemption which can cause other preemptions, increasing the global cost as proposed in [10]. Other solutions aim at controlling the number of preemptions, as presented in [5]. It is worth noting that the preemption cost is not the only cost that must be precisely accounted when dealing with time critical systems. Indeed, the scheduler cost itself must be also precisely accounted. Taking into account the maximum number of preemptions and the scheduler cost, lead usually to increase the WCET up to 50%, for example in the most critical applications of the avionic industry. This pessimism decreases the scheduling ratio and increases the amount of necessary resources. On the other hand, a solution that determines the exact number of preemptions while accounting for the cost of each preemption is proposed in [24]. Unfortunately, this solution assumes that the scheduling algorithm is based on fixed priorities.

Periodic data dependent tasks mean there are precedence constraints between tasks [6] such that a producer task is executed before the corresponding consumer task, and the consumer task must receive the data produced by the producer task. There are two approaches to dealing with precedence constraints. The first one is based on semaphores [12]. A semaphore is allocated to each precedence, and the consumer task must wait for the producer task to release the semaphore before it can start its execution. The second approach is based on the modification of the priorities and the release times of the task [6, 11]. Actually, when dependent tasks have different periods the problem is much more complex than when they have the same period [21]. If the producer task τ_p has a period smaller than the consumer task τ_c, then the latter has to consume in the worst case $n = \lfloor \frac{\tau_c}{\tau_p} \rfloor$ data produced by the producer task. This worst case approach was chosen in [14]. Actually, in general only the last data is consumed since it is considered to be the "freshest" one, as presented in [21]. In contrast, when the producer task has a period greater than the consumer task, the latter has to consume, at worst, n times the same data produced by the producer task. Thus, it is sufficient that the consumer task consumes only one of these data. When tasks are data dependent they have to share some buffer containing the data that may involve priority inversions. For example, a lower priority producer task can block the execution of a consumer task that want to read it while it has a higher priority. In order to avoid this situation, the well known priority inheritance protocol that gives to a task the highest priority of all the tasks which share a data, was proposed in [22]. This protocol holds only for static priority scheduling algorithms. It was extended in [22] by giving an additional priority to every shared data equal to the highest priority of the tasks that share this data. This priority ceiling protocol minimizes the blocking time and prevents deadlocks that may occur when several tasks are mutually waiting for a shared data used by other tasks. For dynamic priority scheduling algorithms, the stack resource policy was proposed in [3].

In order to take into account precisely the preemption and scheduler cost, an offline schedulability analysis that considers the cost of each preemption, is proposed in [19]. Moreover, this analysis allows changes in the priorities of the tasks that are necessary for dependent tasks which involve priority inversions. In this paper after summarizing the principles of this schedulability analysis, we show how the scheduling table it produces is exploited to implement an offline scheduler executed at runtime that is deterministic and thus perfectly suited for time critical embedded systems.

3 Model and Notations

We consider a set Γ_n of n periodic tasks $\tau_i, \forall i \in \{1, 2, \cdots, n\}$ with timing characteristics where:

- r_i^1 is the release time of the first job τ_i^1 generated by the task $\tau_i, \forall i$ and r_i^k is the release time of the k^{th} job;
- T_i is the period of the task τ_i. The k^{th} job τ_i^k is released at $(k-1)T + r_i^1$. A job is said ready for execution from the time $(k-1)T + r_i^1$ until the end of its execution;
- D_i is the deadline of the task indicating that each job τ_i^k has to be scheduled before the time $(k-1)T + r_i^1 + D_i$;
- the set $\Gamma_n(t)$ is the set of all jobs ready for execution at t.

In this paper we consider that the tasks are scheduled on a uniprocessor by the algorithm A. The considered scheduling algorithm A is given and proposing such algorithm is beyond the purpose of this paper. The considered processor has no cache, nor specific internal architecture features. Our hypotheses are not that restrictive as they are usually considered for the design of time critical embedded systems where the time determinism is a key issue.

By preemption cost we understand the time corresponding to the duration necessary to store the context of the preempted task and the duration necessary to restore this context when the preempted task is selected again to resume its execution. Due to its cost, a preemption cost increases the response time of the preempted task that may cause another preemption, and so on. Similarly, by scheduler cost we understand for an online scheduler the time for selecting into the "ready tasks list" the next task to execute and for an offline scheduler the time to read the name of this task in a scheduling table. It is worth noting that in the first case the cost may vary depending on the number of tasks in the list whereas in the second case the cost is constant leading to a more deterministic behaviour.

Definition 1 (**Schedulability interval).** *For a set Γ_n of n tasks scheduled by a scheduling algorithm A, a time interval $[t_1, t_2]$ $(t_1 < t_2)$ is a schedulability interval if $[t_1, t_2]$ is the shortest finite time interval such that if no deadline is missed during this interval then no deadline is missed after t_2.*

Γ_n denotes the set of periodic dependent tasks. The schedulability analysis of Γ_n is achieved on the schedulability interval I_n according to a given fixed or dynamic priority scheduling algorithm, for example Rate Monotonic (RM) or Earliest Deadline First (EDF) [18], to cite only the most famous ones.

3.1 Existing Results Used

The design of our scheduler and its associated schedulability analysis are based on the following existing results:

- **A schedulability interval**

  Given a set Γ_n of n tasks, we use the following schedulability interval denoted I_n and defined as follows:

  $$I_n = [r_{min}, r_{max} + 2H_n] \tag{1}$$

  where

  - r_{min} is the first release time among all jobs of all tasks, $r_{min} = min_{\forall i \in \{1,2,\cdots,n\}}\{r_i\}$;
  - r_{max} is the last release time among the first release time of all jobs of all tasks, $r_{max} = max_{\forall i \in \{1,2,\cdots,n\}}\{r_i\}$;
  - H_n is the least common multiple of the periods of the n tasks.

  The interval I_n is obtained from a schedulability interval $I_n = [r_{min}, t_c + H_n]$ originally proposed in [8] for a set of n periodic data dependent tasks. The time t_c denotes the time from which the schedule repeats indefinitely and its computation is done by an algorithm described in [8]. We do not provide this algorithm here as we only cite it to explain how the interval described by Equation 1 is obtained. As the periodicity of a schedule appears necessarily before $r_{max} + H_n$ [17], then $t_c \leq r_{max} + H_n$. Therefore we replace the interval proposed in [8] by the interval provided in Equation (1).

  It is worth noting that the schedulability interval we propose is actually a schedulability interval if all preemption and scheduler costs are introduced in a periodic manner. In Section 4.5 we provide a theoretical result confirming this hypothesis (see Theorem 1).

- **A priority ceiling protocol**

  Given that our tasks have data dependence constraints thus, according to the classical approach proposed in [7], release times and deadlines of all tasks are modified such that a task τ_j can be executed if and only if each of its predecessors τ_i produces $k_{ij} = \lceil \frac{T_j}{T_i} \rceil$ data, and τ_j does not produce more than $k_{jk} = \lceil \frac{T_k}{T_j} \rceil$ data for each of its successors τ_k. These conditions guarantee that all the data produced are consumed when two data dependent tasks have equal or different periods. However, these conditions do not prevent priority inversions and deadlocks due to data shared by data dependent tasks. This problem is avoided in this paper by choosing for our scheduler the priority ceiling protocol (PCP) [22] which is also known for minimizing blocking times.

4 Schedulability Analysis

The schedulability analysis mimics an offline scheduler using the scheduling algorithm A along the schedulability interval, while accounting for the preemption and scheduler cost. It is presented in detail in [19]. For every scheduler call, the schedulability analysis performs the following steps: it selects a task among the set of ready tasks with the function ϕ, it computes the remaining execution time of the selected task with the function denoted c_i while accounting for the preemption and scheduler cost, it computes the relative deadline of the selected task with the function denoted d_i, it tests the schedulability of the selected task using previous functions whose definitions are recalled in the next sections, and it determines the next scheduler call. As soon as a selected task is not schedulable the set of tasks is not schedulable. If all selected tasks are schedulable the set of tasks is schedulable and the schedulability analysis produces the corresponding scheduling table.

4.1 Task Selection

When the scheduler is called, at time t, a job of a task τ_i will be selected among those belonging to the set $\Gamma(t)$ of jobs ready to be executed at time t. A job τ_i^k of a task τ_i is ready at t if an only if: its first release time occurs before, or at t, and it received all the data produced by its predecessors, and all its successors consumed all the data it produced. The selected task, denoted $\phi(t)$, is the task with the highest priority from $\Gamma(t)$ according to the considered scheduling algorithm A and the priority ceiling protocol as mentioned in Section 3.

4.2 Remaining Execution Time

We denote by $c_i(t)$ the remaining execution time of the job τ_i^k. $c_i(t)$ is defined as the number of time units that the job must still execute at t to complete its execution. If the job τ_i^k is preempted at t, the preemption and scheduler cost is added to the remaining execution time $c_i(t)$ of τ_i^k.

At every release or completion time of a job τ_i^k its remaining execution time $c_i(t)$ is given by:

$$c_i(t) = \begin{cases} C_i & \text{if } (\frac{t-r_i}{T_i}) \in \mathbb{N} \text{ else} \\ c_i(r^-(t)) & \text{if } (\phi(r^-(t)) \neq \tau_i) \text{ else} \\ c_i(r^-(t)) - (t - r^-(t)) & \text{if } (\phi(t) = \tau_i) \vee \\ & \quad ((\phi(t) \neq \tau_i) \wedge \\ & \quad (r^-(t) + c_i(r^-(t)) = t)) \text{ else} \\ c_i(r^-(t)) - (t - r^-(t)) + \alpha \end{cases}$$

where C_i denotes the WCET of τ_i, α the preemption and scheduler cost, and $r^-(t)$ the previous scheduler call.

It is important to note that the WCET of a task is considered here without any approximation of the preemption and scheduler cost since this cost is precisely taken into account with α. However, the WCET includes the precise cost for storing the context when a task is released while preempting another task. In addition, it includes the precise cost of the scheduler, as mentioned in Section 2, which is very simple in our case and can be deterministically and precisely determined. This cost consists in reading, in the scheduling table, the next task to execute when it is released, or when it is resumed if this task were preempted. This is the cost of the interruption routine which implements the time triggered offline scheduler presented in Section 5.2, see Algorithm 2.

The computation of the remaining time $c_i(t)$ correspond to the following four cases:

1. there is a job τ_i^k of a task τ_i which is released at t and thus $c_i(t) = C_i$;

2. during the previous scheduler call the selected job was different from τ_i^k and thus the remaining execution time of τ_i^k does not change $c_i(t) = c_i(r^-(t))$;

3. during the previous scheduler call the selected job was τ_i^k and it is not preempted at t, meaning that τ_i^k is still the selected job at t or, that τ_i completes its execution, thus $c_i(t) = c_i(r^-(t)) - (t - r^-(t))$. That is, the time elapsing between t and the previous value of t corresponding to the execution time of τ_i^k, is subtracted to the previous value of c_i;

4. during the previous scheduler call the selected job was τ_i^k and τ_i^k is preempted at t, it follows that the preemption and scheduler cost α is added to $c_i(r^-(t)) - (t - r^-(t))$.

Of course, such approach accounts for each preemption and scheduler cost which in turn can cause other preemptions, and thus, other preemption and sheduler costs.

In Figure 1 we provide an example of two periodic tasks $\tau_1(2, 2, 6, 6)$ and $\tau_2(0, 3, 8, 8)$ where the timing characteristics between brackets correspond respectively to the first release time, the WCET, the deadline, and the period. We assume that the scheduling algorithm A is Rate Monotonic (RM) and the deadline is equal to the period.

In this example, the offline scheduler is called at times t equal 0, 2, 4, etc., corresponding to the release times and the completion times of both tasks τ_1 and τ_2.

At $t = 0$, τ_2 is released, thus $c_2(0) = 3$. Then, at $t = 2$, τ_1 is released, thus $c_1(2) = 2$ and τ_2 is preempted by τ_1. A time unit (in black) is added to take into account for the cost for restoring the context of τ_2 while the cost for storing the context of τ_2 is assumed to be included in the WCET of τ_1, thus $c_2(2) = (3 - 2 + 0) + 1 = 2$. For the sake of simplicity, in this example we chose one time unit for the cost for restoring the context, but actually it is widely smaller than the WCET. For the same reason we do not indicate the cost for storing the execution context within the WCET of the preempting task. Realistic values of these costs are provided in Section 6. At $t = 4$, τ_1 completes its execution, thus $c_1(4) = 2 - 4 + 2 = 0$ and τ_2 resumes. Since during the previous scheduler call τ_1 was selected, which is different from τ_2, thus $c_2(4) = c_2(2) = 2$, and so on for the other scheduler calls.

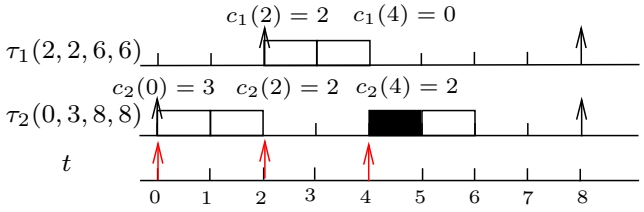


Figure 1: Remaining execution time accounting for the preemption and scheduler cost

4.3 Relative Deadline

We denote by $d_i(t)$ the relative deadline of the job τ_i^k a task τ_i at time t. At every release of τ_i^k, $d_i(t) = D_i$ and $(t - r^-(t))$ is substracted to $d_i(t)$ every time the scheduler is called. In order to do not miss its deadline τ_i must complete its execution before date $t + d_i(t)$.

More formally, the relative deadline $d_i(t)$ is defined as follows:

$$d_i(t) = \begin{cases} D_i & \text{if } (\frac{t-r_i}{T_i}) \in \mathbb{N} \quad \text{else} \\ d_i(r^-(t)) - (t - r^-(t)) & \text{if } r^-(t) + d_i(r^-(t)) > t \\ & else \\ 0 \end{cases}$$

The computation of the relative deadline $d_i(t)$ is obtained by considering the following three cases:

1. the job τ_i^k is released at t;

2. the previous scheduler call added to the previous relative deadline is greater than the present scheduler call, thus the time elapsing between t and the previous value of t, is substracted to the previous value of d_i;

3. the job τ_i^k completes or has already completed at time t.

In Figure 2 we provide an example with one task $\tau_i(0, 3, 8, 8)$. At $t = 0$ the task τ_i is released, for the first time, thus $d_i(0) = 8$. At $t = 2$, $r^-(2) - d_i(r^-(2)) = 0 + 8 > 2$, thus its relative deadline $d_i(2) = 8 - 2 + 0 = 6$ since its previous value was $d_i(0) = 8$.

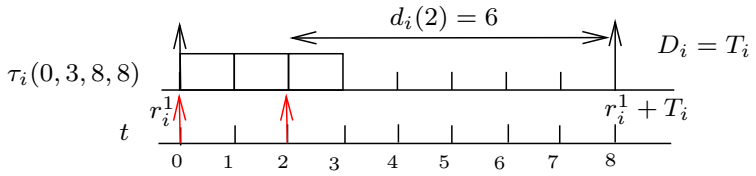


Figure 2: Relative deadline

4.4 Schedulability Condition at t

According to the theorem given in [19], a job τ_i^k of a task $\tau_i \in \Gamma_n$ is schedulable at time t if and only if:

$$\begin{array}{l}(c_i(t) \leq d_i(t)) \wedge ((t \leq r_i) \vee (c_i(r^-(t)) = 0) \vee \\ (\phi(r^-(t)) = \tau_i) \vee ((t - r_i) mod T_i \neq 0))\end{array} \quad (2)$$

where T_i is the period of the task τ_i.

According to Equation 2, the remaining execution time ($c_i(t)$ of τ_i^k at time t is less than or equal to its deadline and one of the four cases may occur:

1. τ_i^k has not yet been released;

2. τ_i^k completes its execution;

3. τ_i^k was the selected task at the previous scheduler call;

4. τ_i does not begin a new job without completing its execution in its previous job.

4.5 Next Scheduler Call

We denote by $r^+(t)$ the next scheduler call and it corresponds to a release or a completion of a task belonging to Γ_n.

$r^+(t)$ is given by:

$$r^+(t) = \begin{cases} t + c_j(t) & \text{if } ((t + c_j(t)) < r(t)) \wedge (\phi(t) = \tau_j) \text{ else} \\ \\ r(t) \end{cases}$$

where $r(t)$ denotes the next release time of a task that belongs to the set $F = \{t \in I_n / \exists(\tau_i, k) \in (\Gamma_n, \mathbb{N}), t = r_i + kT_i\}$ containing the release times of the job released within the schedulability interval I_n of the set of tasks Γ_n. $r(t)$ is the successor element t in F.

In this computation there are two cases:

1. the selected job at t is τ_j^k and its remaining execution time $c_j(t)$ added to t is less than the next release time of a task, i.e. the next scheduler call corresponds to the completion time of τ_j,

2. the next scheduler call corresponds to the next release time of a task.

Another important aspect, when using analyses based on a schedulability interval obtained due to periodicity properties of the schedule, is to ensure that all decisions of the scheduler are not loosing this periodicity. In our case we use a schedulability interval that is originally provided for data dependent tasks (with precedence constraints) and our scheduler introduces the preemption and scheduler cost. In Theorem 1 we provide the

result indicating that, given that all preemption and scheduler costs are added for states of the system that are periodic, then the schedule stays periodic. This result is combined with the fact that the preemption and scheduler cost is added deterministically for the same state of the system by an unique equation (see Section 4.1).

Theorem 1. *The preemption and scheduler cost is introduced periodically by our analysis with a period at most of* H_n.

Proof. It is based on the fact that all decisions are made within the same state of the system that is repeated every H_n [9, 13]. Indeed, as we consider a set of periodic tasks, then all scheduling decisions are made for the state of the system that is repeated with a period H_n and given that our choices are deterministic, then the schedule preserves its periodicity.

4.6 Schedulability Analysis Algorithm

The schedulability analysis applied to the set of task Γ_n is performed according to Algorithm 1. We use a set G which initially is equal to the ordered set F which contains all the release times t of all jobs of all tasks in the interval I_n such that $t_k < t_{k+1}$. The scheduler, at least, will be called at every of these release times in the specific case where all jobs of all tasks are completed when a new job of the same task or of another task, is released. In general jobs are completed before, and when this situation occurs the scheduler is, in addition, called at these completion times. The schedulability analysis iteratively goes through elements t of the set G.

For each iteration, i.e. for each t, it determines, using functions ϕ, c_i, d_i and the schedulability condition (2), as explained in the previous sections, either a job of a task is schedulable or not. If it is the case it proceeds, using the function r^+, to the next scheduler call which is the next t in G or it introduces a new t in G which corresponds to a completed job. As soon as a task is not schedulable the set of task Γ_n is not schedulable. At the end of G, if all jobs of all tasks verify the condition (2) then the set of task Γ_n is schedulable. In this case a scheduling table is produced.

Every line of this table is composed of five columns: t the scheduler call, τ_i^k the name of the selected task to execute (ID), $c_i(t)$ its remaining execution time, $E_i(t)$ the whole or a part of the remaining execution time $c_i(t)$ to be executed by the task (DURATION) and the status (STATUS) of the task. When a job of a task starts, its remaining execution time $c_i(t) = C_i$, then this value decreases every time the task is resumed after a preemption taking into account the preemption and scheduler cost. Actually, the task must be executed $E_i(t)$ which can be equal to either $c_i(t)$ when the task is preempted or only a part of $c_i(t)$ when the task is resumed. Of course, $E_i(t)$ corresponds to the duration elapsing between two consecutive calls of the scheduler. We shall use this term "duration" afterwards. The status may take the following values: 1: start of a job of a task (START), 0: resume of job of a task (RESUME), -1: "idle task" (IDLE). The "idle task" corresponds to the time elapsing between the completion of a job of a task and the release of the next job of the same task or a job of another task, i.e. when the processor is idle. On the other hand, when the task is completed it sets the flag COMPLETED at *true* that the scheduler will read afterwards. An example of such a scheduling table is given for a set of three tasks in Table 2 of Section 6.2.1.

The complexity of this schedulability analysis algorithm is, in the worst case, equal to $O(2mn)$ with m the number of different release times in the schedulability interval I_n associated to the set of n tasks Γ_n.

It is worth noting that this schedulability analysis is sustainable according to the WCET as demonstrated in [19]. That is, even though some tasks have execution times shorter than their WCET then the set of tasks remains schedulable. This is an important property used to validate conclusions given for the schedule measured at runtime with the corresponding time triggered offline scheduler.

5 Time Triggered Offline Scheduler

5.1 Time Triggered Approach

When dealing with schedulers at runtime, there are two approaches for triggering the tasks they manage [16]. In the usual event triggered (ET) approach the scheduler, triggered by external interruptions, selects according to an online scheduling algorithm, the next task to execute among the "ready tasks list". In the time triggered (TT) approach the scheduler, triggered at predefined times, finds the next task to execute in a scheduling table built offline. Defining these times requires, a complete understanding of the system and of the environment it will operate in. Since we perform an offline schedulability analysis which produces a scheduling table, the TT approach is the best suited for implementing our scheduler. This approach compared to ET online scheduler

Algorithm 1 Schedulability analysis

1: $t \leftarrow r_{min}$
2: $G \leftarrow F$
3: $schedulable \leftarrow true$
4: **while** $(t < (tc + H_n)) \wedge (schedulable = true)$ **do**
5: Compute $\phi(t)$
6: $i \leftarrow 1$
7: **while** $(i \leq n) \wedge (schedulable = true)$ **do**
8: **if** $(t \geq r_i)$ **then**
9: Compute $c_i(t)$
10: Compute $d_i(t)$
11: **if** $((c_i(t) > d_i(t)) \vee$
$((t > r_i \wedge (c_i(r^-(t)) > 0) \wedge (\phi(r^-(t)) \neq \tau_i) \wedge ((t - r_i) mod T_i = 0))$ **then**
12: $schedulable \leftarrow false$
13: **end if**
14: **end if**
15: $i \leftarrow i + 1$
16: **end while**
17: $t \leftarrow r^+(t)$
18: $G \leftarrow G \cup \{r^+(t)\}$
19: **end while**

has the following main advantages. It prevents from exploring, online, the "ready tasks list" whose length varies according to the scheduler calls, in order to select the next task to execute. Obviously, these variations involve variations in the scheduler cost. In addition, it prevents from managing online priority inversions and deadlocks, which increases the variation of the scheduler cost, since they have been taken into account during the offline schedulability analysis. Consequently, as already mentioned in the schedulability analysis, and as it will be shown afterwards at runtime, the time triggered offline scheduler is greatly simplified and deterministic compared to usual online scheduler, since its cost does not vary and is easily determined.

Among the TT schedulers, the best known are those that are called periodically. A periodic timer calls the scheduler at a predefined period that is, at best, the greatest common divisor of the task periods to prevent release and completion time misses. The main drawback of this approach is that the scheduler may be called more than necessary. The second kind of TT scheduler is called only at appropriate times [15, 20]. Actually, these times are those stored in the scheduling table.

5.2 Offline Scheduler

With such a scheduler, at runtime no task need to be selected since this selection has already been performed, offline, during the schedulability analysis. However, some actions are necessary in order to actually execute the tasks. Of course, we use the second kind of TT scheduler which uses the scheduling table produced by the offline schedulatility analysis.

In each line of this table, detailed in Section 4.6, only the time when the scheduler is called, the name of the selected task and the duration, are used by the scheduler. Actually, the duration elapsing between two consecutive calls of the scheduler is obtained by substracting two consecutive times when the scheduler is called. The duration is used as an initial value of a timer which is decremented at every cycle of its clock and as soon as the timer reaches zero an interruption routine is called. This routine is composed of two parts, the first part implements the scheduler whereas the second part, at the end of the routine, stores the context of the previous task, restores the context of the next task and finally launches the next task. The interruption routine is based on Algorithm 2. The first part uses two tables as input: the scheduling table mentioned just above and an additional task table that holds the context of every task. The latter table holds also the context of a specific task, called "idle task". This specific task is an infinite loop running the processor when no task runs it. Once a timer interruption occurs, the interruption routine is called which immediatly loads the timer with the current duration until the next call of the interruption routine. It then updates the previous and current indexes of the scheduling table (previous index becomes current index and current index becomes next) while next index is updated at the end of the interruption routine (next index is incremented). Then, it reads the scheduling table line at the current index. Now, using the status, if the next task starts a new job, and if the previous job of the next task is not completed then the next task prints "Deadline Missed", otherwise the next task is a new job

of the same task. Identically, using the status, if the next task is resumed because it was preempted by another task, and if the previous job of the next task is completed then, the next task is an "idle task", otherwise the task is resumed and it continues the execution of its previous job.

For the interruption routine running on the ARM Cortex-M4 we give, in annexe A, the flowchart of Algorithm 2, the C code for the first part (scheduler) and the assemby code of the second part (store and restore of the context) at the end of the routine.

Algorithm 2 Interruption routine $INT\_HANDLER$

```
1: /* i_prev, i and i_next are extern variables initialized to 0
2: /* T_SCHED data structure containing: ID of the task, DURATION, STATUS (START, RESUME) */
3: INPUT1 : scheduling table T_SCHED[SIZE_T]
4: /* T_TASKS data structure containing: address of the CODE to execute, COMPLETED (TRUE, FALSE) */
5: INPUT2 : tasks table T_TASKS[TASKS_NUMBER]
6: /*update scheduling table indexes i, i_prev, and i_next*/
7: i_prev ← i
8: i ← i_next
9: /*load the timer TIMER with time duration stored in T_SCHED[i].DURATION and start counting*/
10: LOAD_TIMER(TIMER, T_SCHED[i].DURATION)
11: START_COUNT(TIMER)
12: /*read the ID of the next task to execute from the scheduling table T_TASKS[T_SCHED[i].ID] ...*/
13: next_task = T_TASKS[T_SCHED[i].ID]
14: /*if there is a new job of the task T_TASKS[i]...*/
15: if (T_SCHED[i].STATUS == START) then
16:    /*...we verify if the task has completed its previous job before beginning a new one*/
17:    if (T_TASKS[T_SCHED[i].ID].COMPLETED == FALSE) then
18:       /*T_TASKS[T_SCHED[i].ID] missed its deadline...*/
19:       next_task = printDeadlineMissed(T_TASKS[T_SCHED[i].ID])
20:    end if
21:    /*else if the status is resumed T_SCHED[i].STATUS == RESUME, since task was preempted previously */
22: else if (T_SCHED[i].STATUS == RESUME) then
23:    /*... if task is completed call the idle */
24:    if (T_TASKS[T_SCHED[i_prev].ID].COMPLETED == TRUE) then
25:       next_task = idle_task
26:    end if
27: end if
28: if i == SIZE_T − 1 then
29:    i_next ← I_PERM
30: else
31:    i_next ← i + 1
32: end if
33: /* this part is written in assembly code */
34: /* CONTEXT contains the context of the task */
35: STORE_CONTEXT(T_SCHED[i_prev].CODE, T_TASKS[T_SCHED[i_prev].ID].CONTEXT)
36: RESTORE_CONTEXT( next_task.CONTEXT)
37: EXECUTE( next_task.CODE)
38: /* the interruption routine ends */
39: RETURN_FROM_INTERRUPT()
```

6 Performance Evaluation

6.1 Experimental Conditions

In order to evaluate the offline scheduling analysis and the time triggered offline scheduler proposed in the previous sections, we sought a processor that complied with assumptions we made, i.e. neither cache nor complex pipeline or specific internal architecture features.

The ARM Cortex-M4 processor with no data and instruction caches, and with a simple 3-stage pipeline, was a good candidate for our evaluation. With this processor a program executed several times with the same input

data will produce the same output dada, as soon as the initial state is the same. This requires that registers and memories are always initialized in the same way. Therefore, the programs will be executed deterministically. This is a crucial issue when aiming at accurately measure the execution time of programs, not only the program corresponding to the context storing and restoring due to preemptions as well as the scheduler, but also the programs associated to tasks. Indeed, we need to measure accurately the WCET of the tasks without any approximation, whether they are real tasks provided by industry or synthetic tasks (for example an empty loop consuming time according to the number of times it is executed). The ARM Cortex-M4 also provides an integrated debug unit, very useful for monitoring the execution of the code, and for introducing breakpoints between the start and the end of a program to measure its duration. There are two modes of execution in this processor, a privileged one with a seperated stack pointer and set of registers that we use for the scheduler context, and an unprivileged one that we use for the tasks.

In addition, the ARM Cortex-M4 is a widely spread and very popular processor in the embedded world. There are many chip vendors offering microcontrollers based on that processor including usual peripherals for communicating with the outside (USB, ethernet, etc.), accurate timers, flash storage, memory, etc.

For our evaluation, we chose the LPC4088 microcontroller from NXP based on the ARM Cortex-M4. It is available inside a development board called EA LPC4088 QuickStart Board from Embedded Artists. The development board is shown in Figure 7 where the LPC4088 microcontroller can be easily seen in the center of the board. The LPC4088 microcontroller provides a set of four hardware timers and a clock configuration unit to set their running periods. We use a common clock for the CPU and timers in order to avoid drifts between them. We use only two of these timers, one for the time triggered scheduler as explained in Section 5, configured in high priority interrupt mode, and one for time measurement purposes. The latter is used for reading elapsed times and then are reset to be used later on. Notice that timers, and in general peripherals, use a dedicated peripheral bus which helps to reduce access delays when setting the timers.

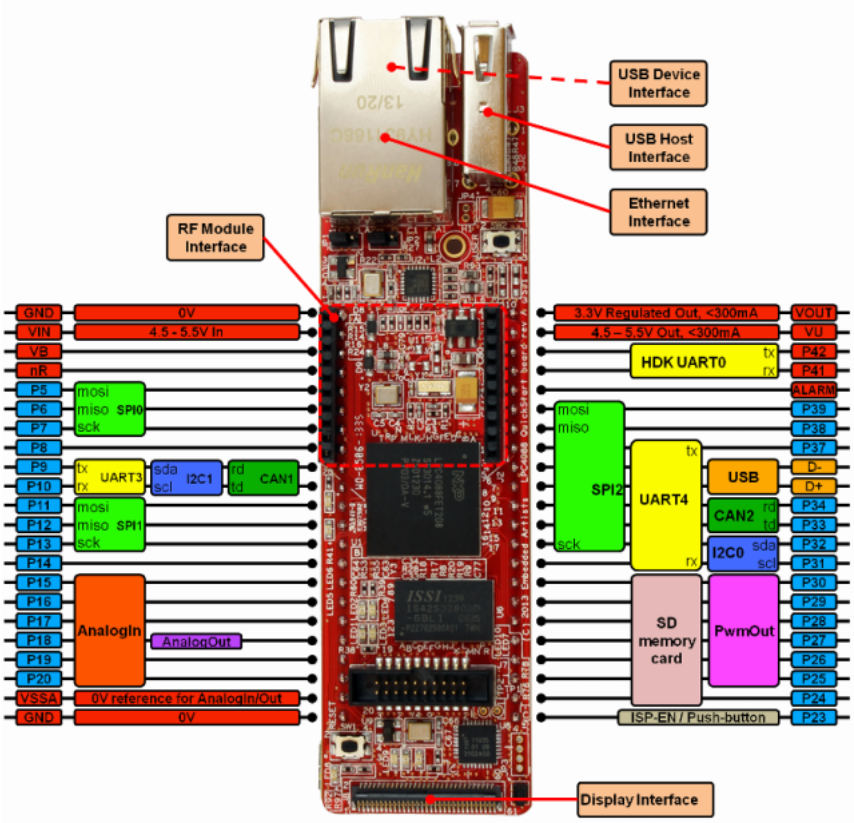


Figure 3: Overwiew of the development board using an ARM Cortex-M4 processor

6.2 Experiments

In order to illustrate the benefit of accounting for the preemption and scheduler cost, we carry out two experiments. The first experiment illustrates the main principles of the approach accounting for the preemption and scheduler cost. It shows that the runtime schedule of a task set measured on the ARM Cortex-M4 processor corresponds to the schedule of the same task set predicted by the offline schedulability analysis. The second experiment goes into deeper details. It shows that, if the preemption and scheduler cost is incorrectly estimated, this can leads to missed deadlines in the runtime schedule whereas the predicted schedule is correct.

In both experiments, for a given set of tasks, we carry out a schedulability analysis which generates a scheduling table. For the sake of simplicity, we chose RM as scheduling algorithm A for this offline schedulability analysis.

In the first experiment the scheduling table corresponds to the predicted schedule which is presented afterwards with a "predicted timing diagram". Then, this scheduling table is used by a time triggered offline scheduler, as presented in Section 5.2, to run the task set on the ARM Cortex-M4 processor of the LPC4088 microcontroller located on the development board. This scheduler written in C and assembly codes implements the Algorithm 2 as an interruption routine. When the task set is running on the ARM Cortex-M4 processor, a logging code allows the measurements of release, completion, preemption and resume times for each task. These measures of time correspond to the runtime schedule which is presented afterwards with a "measured timing diagram". Finally for the same task set, we compare the predicted timing diagram produced by the schedulability analysis to the measured timing diagram obtained on the ARM Cortex-M4.

In the second experiment we use a scheduling simulation tool (SAS) that produces the corresponding timing diagrams. The tool has two modes: a mode which does not takes into account the preemption and scheduler cost, and a mode which takes into account the preemption and scheduler cost. The timing diagram corresponding to the latter mode is directly equivalent to the scheduling table produced by the offline schedulability analysis, that is, to the predicted timing diagram. Thus, we can compare the predicted and the measured timing diagrams which both do not take account the preemption and scheduler cost, with the predicted timing diagram which takes account the preemption and scheduler cost.

The preemption and scheduler cost is measured on the ARM Cortex-M4. We choose to measure this cost as a number of clock cycles of the processor to be independent of the clock value. We measure the execution time of the first part of the interruption routine and obtain 161 cycles for executing the scheduler. We obtain 61 cycles for storing the context, restoring the context and launching the task corresponding to the second part of the interruption routine. Thus, the preemption and scheduler cost is equal to 222 clock cycles. This is this value which is used as the parameter α mentioned in Section 4 presenting the schedulatiblity analysis. On the other hand, this value is also taken into account when considering the WCET of every task. Indeed, the scheduling table generated by the schedulability analysis contains for every scheduler call the name of the selected task and the duration this task must be executed. In order to take into account the preemption and scheduler cost it is included into this duration.

6.2.1 First Experiment

In this experiment we compare, using timing diagrams, the schedule predicted by our offline schedulability analysis with the schedule measured on the ARM Cortex-M4 while accounting for the preemption and scheduler cost and taking into consideration the impact of data dependences. We consider a set of three dependent tasks with the corresponding timing characteristics r_i^1, C_i, D_i and T_i given in Table 1. Values in this table and in the other tables of timing characteristics, as well as in the scheduling tables, correspond to a number of time units. We take the scheduler and preemption cost as time unit. This means that a time unit is equal to 222 clock cycles of the ARM Cortex-M4. The data dependence constraints are described by the data dependence graph given in Figure 4.

Scheduling table

Using the schedulability analysis presented in Section 4 applied to the schedulability interval $I_3 = [r_{min}, r_{max} + 2H_3] = [0, 10 + 2 * 24] = [0, 58]$ with $H_3 = ppcm(6, 12, 24) = 24$ we obtain the scheduling table given in Table 2. We recall that in this table, every line is composed of five columns: t, τ_i, $c_i(t)$, $E_i(t)$ and *status*.

According to the proposed offline scheduling approach, let us comment Table 2. At $t = 0$, the first job of task τ_2 is released and started, thus $c_2(t = 0) = 5$ equals its WCET. It only executes $E_2(0) = 2$ of 5 units since it will be preempted by τ_1 which has a higher priority than τ_2 (RM scheduling algorithm). Thus, the first job of τ_1 is released and started at $t = 2$ and it executes $c_1(t = 2) = E_1(2) = 2$ equal to its WCET and is completed at $t = 4$. At the next scheduler call $t = 4$, τ_2 is resumed and $c_2(t = 4) = E_2(4) = 3+1 = 4$ instead of 3 because one

time unit is added to $c_2(t = 4) = 3$ corresponding to the preemption and scheduler cost, since we take the value of this cost as time unit. Thus, the first job of τ_2 which is executed 6 units rather than 5 units, is completed at $t = 8$ which corresponds to the next scheduler call. At this next scheduler call $t = 8$, the second job of task τ_1 is released and started, thus $c_1(t = 8) = E_1(8) = 2$ equals its WCET and because it has the highest priority τ_1 is completed at $t = 10$ where the first job of τ_3 is released and started. It executes $c_3(t = 10) = E_3(10) = 3$ equals to its WCET. Thus, the first job of τ_3 is completed at $t = 13$ relinquishing the processor which becomes idle (status IDLE) until the third job of τ_1 is released and started at $t = 14$. Identically, the processor becomes idle from $t = 16$ to $t = 20$ and from $t = 41$ to $t = 44$. Then, the fourth job of τ_1 is released and started at $t = 20$ and is completed at $t = 22$. The second job of τ_3 is released and started at $t = 22$ and is completed at $t = 25$. In the meantime the second job of τ_2 is released at $t = 24$ but it is started at $t = 25$ since its priority is lower than the priority of the second job of τ_3. At $t = 26$ the fifth job of τ_1 is released and started, preempting the second job of τ_2 whose priority is lower than the priority of τ_1. It is worth noting that it is the preemption of the task τ_2 by the task τ_1 at $t = 26$ which will cause its second preemption at $t = 32$, and that it is due to the fact that the preemption and scheduler cost is taken into account. Indeed, when τ_2 is preempted at $t = 26$ if the preemption and scheduler cost was not taken into account the task τ_2 would have been completed at $t = 32$ instead of $t = 36$. It would have been preempted only once rather than twice, at $t = 26$ and $t = 32$. And so on until the end of the scheduling table.

Tasks	r_i^1	C_i	D_i	T_i
τ_1	2	2	6	6
τ_2	0	5	24	24
τ_3	10	3	12	12

Table 1: Task set of first experiment

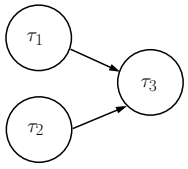


Figure 4: Data dependence graph of first experiment

On the other hand, we can see in the scheduling table that both data transferts and mutual exclusion constraints are taken into account by the schedulability analysis. Indeed, at $t = 34$ the sixth job of task τ_1 is completed and the third job of task τ_3 is released. However, the task τ_3 is not started even though it has a higher priority than the task τ_2, leading to a priority inversion. This delay occurs because τ_2 must be executed once for two executions of τ_3. That allows the task τ_3, executing at period 12, to consume twice the same data produced at period 24 by the task τ_2.

Inversely, at $t = 38$ the seventh job of task τ_1 is released, this task is not started even though it has a higher priority than task τ_3, because τ_3 must be executed twice for one execution of τ_1. This delay allows the task τ_1, executing at period 6, to produce two data while the task τ_3 consumes these two data at period 12.

Finally, an important feature of the proposed approach concerns the transient and the permanent phases of the scheduling table. For the given task set, the transient phase of the scheduling table starts from $t = 0$ to $t = 25$. Then, the permanent phase starts from $t = 25$ to $t = 58$. That means that at runtime the transient phase will is executed once whereas the permanent phase is indefinitely executed.

Predicted timing diagram

The predicted timing diagram given in Figure 5 is an exact graphic representation of the scheduling table given in Table 2. The horizontal axis (x-axis) depicts the time from $t = 0$ to $t = 50$ corresponding to a representative interval of the scheduling table. The vertical axis (y-axis) depicts the different tasks with different colors. The priority of the tasks increases from bottom to top. Since we use RM as scheduling algorithm in the offline schedulability analysis, task "tau1" (τ_1) has a higher priority than task "tau2" (τ_2) which, in turn, has a higher priority than task "tau3" (τ_3). For every task, the low state denotes that the task is not executed during this time interval whereas the high state denotes that it is executed.

In accordance with comments given previously about the scheduling table, the specific times correspond-

t	$\phi(t) = \tau_i$	$c_i(t)$	$E_i(t)$	$Status$
0	τ_2	5	2	1
2	τ_1	2	2	1
4	τ_2	4	4	0
8	τ_1	2	2	1
10	τ_3	3	3	1
13	$idle$	1	1	-1
14	τ_1	2	2	1
16	$idle$	4	4	-1
20	τ_1	2	2	1
22	τ_3	3	2	1
24	τ_3	1	1	1
25	τ_2	5	1	1
26	τ_1	2	2	1
28	τ_2	5	4	0
32	τ_1	2	2	1
34	τ_2	2	2	0
36	τ_3	3	2	1
38	τ_3	1	1	1
39	τ_1	2	2	1
41	$idle$	3	3	-1
44	τ_1	2	2	1
46	τ_3	3	2	1
48	τ_3	1	1	1
49	τ_2	5	1	1
50	τ_1	2	2	1
52	τ_2	5	4	0
56	τ_1	2	2	1
58	τ_2	2	4	0

Table 2: Scheduling table

ing to the start, the preemption, the resumption and the completion of each task, while taking into account the preemption and scheduler cost, as well as the specific times corresponding to tasks delayed due to data dependences, are highlighted by vertical lines on the predicted timing diagram given in Figure 5.

Measured timing diagram

In the measured timing diagram given in Figure 6 the x-axis depicts the real time from $t = 0$ to $t = 12400$ where t represents a numbers of clock cycles of the ARM Cortex-M4, always to be independent of the value of its clock. This interval corresponds to the same representative interval of the scheduling table that we used for the predicted timing diagram. Identically to the predicted timing diagram, the y-axis depicts the different tasks with same colors. The priority of the tasks increases from bottom to top. In addition, there are two other specific tasks called "sched" and "idle". The latter task corresponds to all idle times of the processor, i.e. when a job of a task is completed before the next job of the same task or of another task. The "sched task" executes the scheduler, i.e. the interruption routine presented previously in Section 5.2. That is, it reads in the scheduling table the next task, stores and restores the context and, of course, launches the next task. In other words, this specific "sched task" corresponds to the preemption and scheduler cost for all the tasks. Overall, this timing diagram diagram corresponds to the measured schedule.

We can observe on the measured timing diagram that, at $t = 222$, the first job of τ_2 is released after the execution of the scheduler corresponding to its first call. In order to be precise it is worth noticing that, on the diagram the duration of the scheduler is equal to 161 cycles whereas the first job of τ_2 starts at $t = 222$ leading to an empty space in the diagram. That is consistent with values given in the introduction of Section

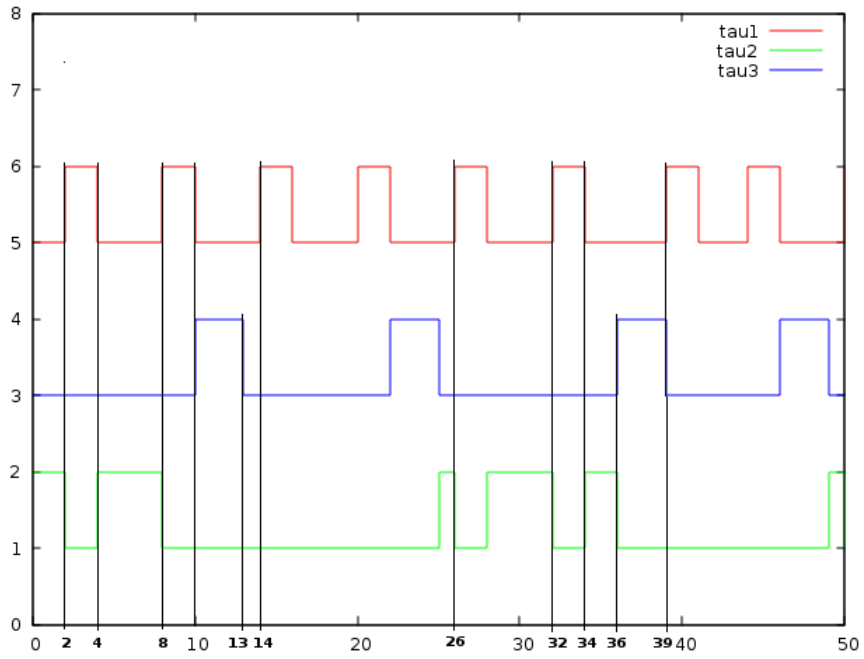


Figure 5: Predicted timing diagram

6.2 that is, 161 cycles for the cost of the scheduler and 61 cycles for storing the context, restoring the context and launching the task. The latter cost is not drawn on the diagram since it corresponds to an assembly code which cannot be logged. When carefully looking at the measured timing diagram, such empty spaces can be observed every time the scheduler task is completed. Then, at $t = 772$ the first job of τ_1 is released after the execution of the scheduler (161 cycles) and an empty space (61 cycles), and it preempts τ_2. Then, the first job of τ_2 which was preempted by τ_1 is resumed at $t = 1286$ always after an execution of the scheduler (161 cycles) and an empty space (61 cycles). Then, the second job of τ_1 is released at $t = 2264$ always after an execution of the scheduler and an empty space. Further, we can observe that at time $t = 2812$ the first job of task τ_3 is released and it is completed at $t = 3164$, that is, before the next call of the scheduler, leading to an "idle time" until $t = 3534$ where the scheduler is executed before the third job of τ_1 which is released at $t = 3756$. And so on.

Comparison

Considering that timing scales are different, the measured timing diagram is identical to the predicted one. Indeed, we can find the same sequence of release, completion, preemption and resume times for every job of every task, as in the predicted timing diagram. Moreover, we can find for every of these times an occurence of the "sched task" corresponding to a scheduler call. Similiarly, we can find an "idle task" corresponding to idle times of the processor when a task completes before the release of the same task, or of another task. In order to be precise, it is worth mentioning that some tasks may complete before the expected time. This is due to the fact that synthetic tasks that we use may have an execution time which is smaller, but under any circumstances larger, than the corresponding execution time put in the scheduling table produced by the offline schedulability analysis of the given task set. This behaviour is consistent with the sustainability property of our approach mentioned in Section 4.4.

This comparison shows that, for a given task set, the result of our offline schedulability analysis accounting for the preemption and scheduler cost precisely measured on the ARM Cortex-M4, is correctly implemented on this processor running a time triggered scheduler. In other words, the task set executes at runtime exactly as it was predicted by the scheduling table produced by the offline schedulability analysis.

6.2.2 Second Experiment

Considering the principles presented in the previous experiment we go further in detail to show that neglecting the preemption and scheduler cost may lead to deadline misses whereas classical schedulability analyses predict a correct runtime schedule without any dealine miss. In order to accurately depict these details we use more

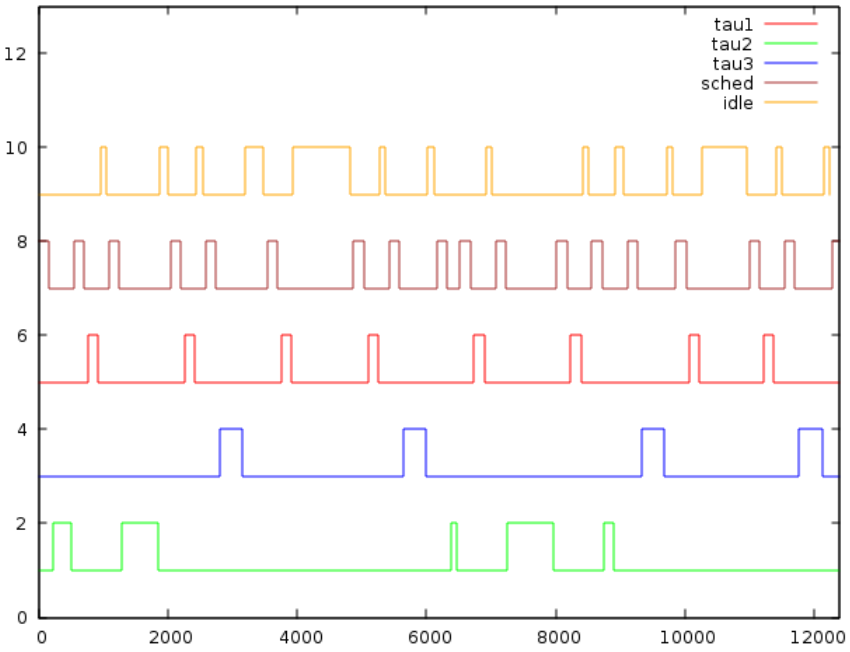


Figure 6: Measured timing diagram

compact timing diagrams that allow to exhibit more information in the same page width.

We provide three scenarios in order to illustrate the impact of the preemption and scheduler cost on the schedulability of tasks, as follows:

- even a job of the highest priority task may miss its deadline due to this cost;
- a (less counter-intuitive) job of a less priority task may also miss its deadline;
- introducing this cost may cause new preemptions by increasing the response time duration and these new preemption and scheduler costs will cause other deadline misses.

In order to cover all the task set cases, the two first scenarios are applied to a first set of independent tasks whereas the third scenario is applied to a second set of dependent and independent tasks. Note that the proposed offline schedulability analysis producing a scheduling table and the time trigger offline scheduler using this scheduling table, works alike on independent and dependent tasks.

In figures presented afterwards, the measured timing diagrams obtained at runtime are such that the first row shows the preemption and scheduler cost in black along with the "idle task" in grey, while the following rows show the scheduling of the tasks ordered from the highest to the lowest priority. In order to easily modify its WCET, each task is built as a loop, with different stop condition values.

The timing characteristics of the first task set are given in Table 3. It contains three independent tasks t_1, t_2 and t_3. We use our offline schedulability analysis with RM as scheduling algorithm like in the previous experiment. The corresponding timing diagram produced by the scheduling simulation tool in the mode that does not take into account the preemption and scheduler cost, is given in Figure 7. Because of its greatest period, t_3 is assigned the lowest priority by RM and therefore is preempted five times during each job by the two higher priority tasks t_1 and t_2. t_2 is the middle priority task and is preempted only one time during each job. Consequently t_3 and t_2 must pay respectively five and one preemption and scheduler costs. The highest priority t_1 is not preempted but must pay the preemption and scheduler cost each time it preempts another task, whether t_2 or t_3.

• *First scenario high priority task deadline miss:* we show that even a high priority task t_1 that is not preempted can miss its deadline if we do not precisely account for the preemption and scheduler cost. In figure 7 we show the simulated timing diagram without accounting for the preemption and scheduler cost. Thus, transitions between tasks are assumed ideal and the taskset is deemed schedulable by the scheduling simulation

Tasks	r_i^1	C_i	D_i	T_i
t_1	30	20	50	50
t_2	20	25	100	100
t_3	0	100	300	300

Table 3: Task set 1 scenario 1 and 2 timing characteristics

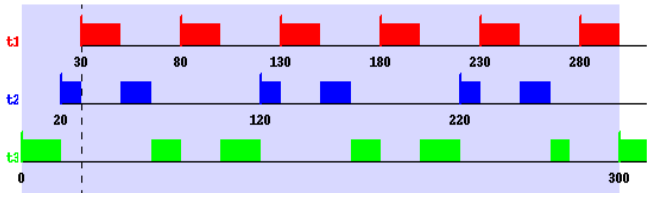


Figure 7: Scenario 1 simulated timing diagram without the preemption and scheduler cost

tool. A scheduling table is produced by our offline schedulability analysis and used to run the task set on the ARM Cortex-M4. This scenario is recorded with the logging tool on the real platform which produces the runtime measured diagram shown in figure 8. We see that the high priority task t_1 misses its deadline because the preemtion and scheduler cost consumes a part of its time budget. Note that the lower priority tasks t_2 and t_3 do not miss their respective deadlines.

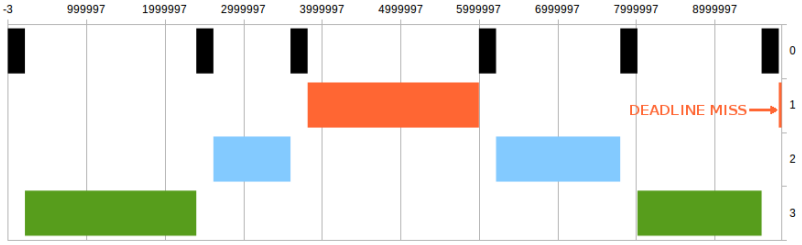


Figure 8: Scenario 1 measured timing diagram

Now, we use the scheduling simulation tool in the mode that takes into account the preemption and scheduler cost. It produces the timing diagram given in Figure 9 where t_1 does not miss its deadline and a new job of t_1 is released at time 80. In this case the measured timing diagram obtained at runtime should be identical to the predicted one as explained in the first experiment presented in Section 6.2.1.

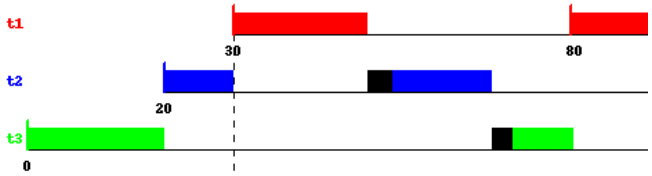


Figure 9: Scenario 1 simulated timing diagram with the preemption and scheduler cost

• *Second scenario low priority task deadline miss:* here we keep the same task set and scheduling table generated in the first scenario without acounting for the preemption or scheduler cost, and to avoid that task t_1 miss its deadline at runtime on the ARM Cortex-M4 like in the first scenario, we slightly decrease its effective execution time to fit in the available budget.

We focus on the last part of figure 7, shown in figure 10. When t_3 completes its job, there is still available time before t_1 is released. However, since t_3 is preempted five times, it pays five preemption and scheduler costs that are ignored in figure 7. When these costs are sufficiently large a deadline miss occurs as it is depicted in the runtime measured timing diagram 11.

As shown on the zoomed offline timing diagram given in figure 12 where the preemption and scheduler costs are taken into account, t_3 misses its deadline, hence by accounting for these costs we were able to predict what really happened on the measured diagram in figure 11. Consequently, the task set is a priori not schedulable when we take into account the preemption and scheduler cost, that better describes the reality.

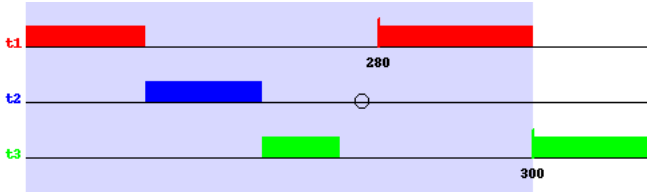


Figure 10: Scenario 2 simulated timing diagram without the preemption and scheduler cost

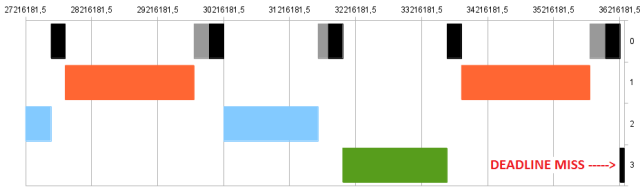


Figure 11: Scenario 2 measured timing diagram

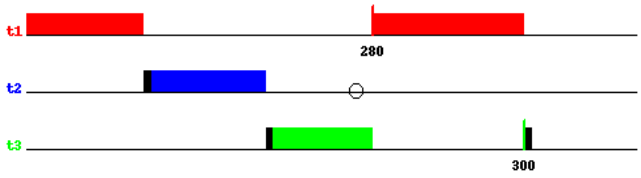


Figure 12: Scenario 2 zoomed simulated timing diagram with the preemption and scheduler cost

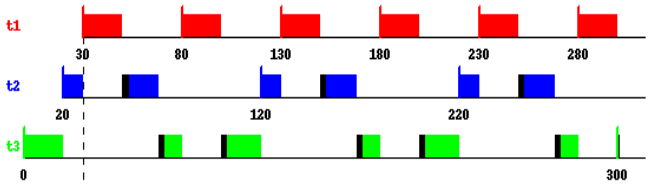


Figure 13: Scenario 2 simulated timing diagram with the preemption and scheduler cost

The complete offline timing diagram with preemption and scheduler costs is shown in figure 13.

The timing characteristics of the second task set are given in table 4. Some tasks of this set have data dependence contraints described by the data dependence graph shown in Figure 14. It contains three dependent tasks t_1, t_2 and t_3 and one independent task t_4. We use our offline schedulability analysis, also with RM as

Tasks	r_i^1	C_i	D_i	T_i
t_1	30	50	250	250
t_2	120	75	250	250
t_3	200	20	250	250
t_4	0	500	3000	3000

Table 4: Task set 2 scenario 3 timing characteristics

scheduling algorithm. The corresponding timing diagram produced by the scheduling simulation tool in the mode that does not take into account the preemption and scheduler cost, is given in Figure 15.

Because of its greatest period, t_4 is the lowest priority task. The release dates of higher priority tasks are offseted to avoid mutual preemption, but they have to pay the preemption and sceduler cost at each release and when they preempt t_4. Due to this configuration (higher priority tasks are more frequently released than the low priority task t_4) several preemptions occurs for one job of t_4. Such situation occurs when for instance a background task is frequently preempted by sensor, actuator, and control tasks.

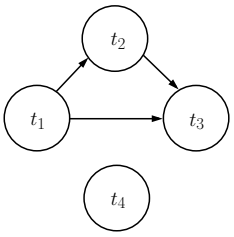


Figure 14: Task set 2 scenario 3 data dependence graph

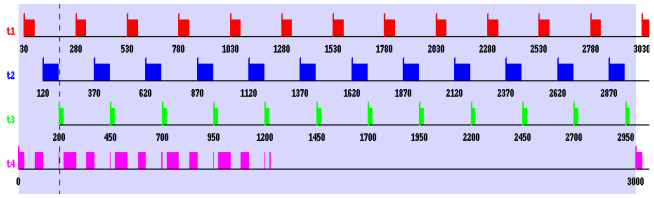


Figure 15: Scenario 3 simulated timing diagram without the preemption and scheduler cost

• *Third scenario where preemptions produce other preemptions:* in this scenario we show that when several preemptions occur during a job of the low priority task t_4, the preemption and scheduler costs can sufficiently delay the task and cause additional preemptions in a cascading effect, resulting in a deadline miss. Here the task t_4 is preempted fifteen times according to the offline schedulability analysis shown in the timing diagram given in Figure 15. Thus, it pays a large value of preemption and scheduler cost that causes the task execution to continue beyond its offline completion time at 1230 pointed by the circle in the zoomed simulated timing diagram with preemption and scheduler cost given in Figure 19, and pointed at position ① on the zoomed measured timing diagram in Figure 18. Moreover, the task t_4 proceeds until $r_1^6 = 1280$ the sixth release time of the higher priority task t_1. This causes a new preemption for t_4 shown at position ② in the zoomed measured timing diagram 18, and because there is no scheduling table line afterwards to resume from this additional preemption, as shown after position ③ in the zoomed measured timing diagram 18, t_4 misses its deadline. Here, although the runtime utilization factor $UE = 0.7733$ would be not very close to one, a deadline miss occurs because of a lack in the scheduling table of the t_4 resume line, corresponding to this additional preemption.

As shown on the simulated timing diagram given in Figure 19 where the preemption and scheduler cost is taken into account, t_4 does not miss its deadline, whereas it misses its deadline in the runtime measured timing diagram given in Figure 18.

To sum up the second experiment, the proposed approach avoids three different deadline miss scenarios by introducing in the schedulability analysis the preemptions and scheduler cost. This cost, used in the schedulability analysis, is precisely measured on the ARM Cortex-M4 where task sets are executed at runtime. For that

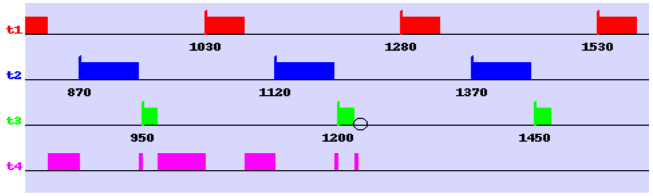


Figure 16: Scenario 3 zoomed simulated timing diagram

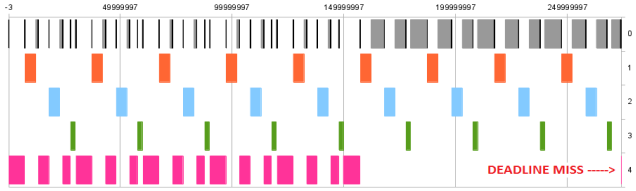


Figure 17: Scenario 3 runtime measured timing diagram

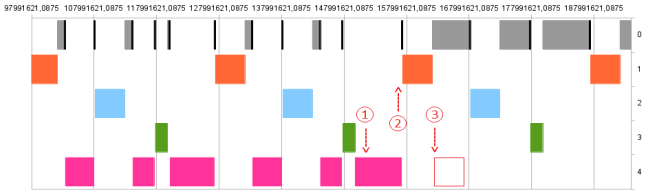


Figure 18: Scenario 3 zoomed runtime measured timing diagram

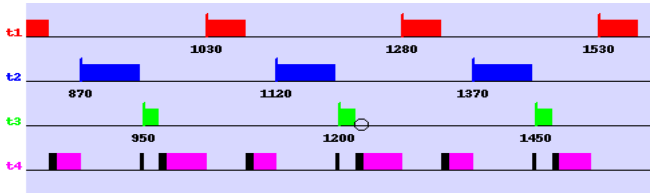


Figure 19: Scenario 3 simulated timing diagram with the preemption and scheduler cost

purpose we use a methodology composed of three steps. In the first step we provide with a usual simulation tool the scheduling without taking into account the preemption and scheduler cost. Then, in the second step we compare these results with the scheduling measured by running the task set on the ARM Cortex-M4, and we observe deadline misses. Finally, we show that the improved offline schedulability analysis taking into account the preemption and scheduler cost, avoids deadline misses of scenario one and three, and predicts the deadline miss of scenario two.

7 Conclusion and Future Work

We proposed an offline schedulability analysis for data dependent periodic tasks which precisely accounts for the preemption and scheduler cost. The scheduling table produced by this analysis is exploited in a time triggered offline scheduler which guarantees that no deadline misses occur in accordance with the schedulability analysis. We evaluated this scheduler on an ARM Cortex-M4 bare metal uniprocessor, and showed that it is able to schedule correctly set of tasks that miss their deadline when the preemption and scheduler cost is neglected. Therefore, such a scheduler is perfectly suited for time critical embedded systems.

As future work, we intend to extend our offline schedulability analysis accounting for the preemption and scheduler cost for uniprocessors with more complex internal architecture, and for processors with several cores.

A Annexe

In this annexe we present in Figure 20 the flowchart of the interruption routine composed of two parts, the part which implements the scheduler and, at the end of the routine, the part which stores and restores the context of the task and finally launches the task.

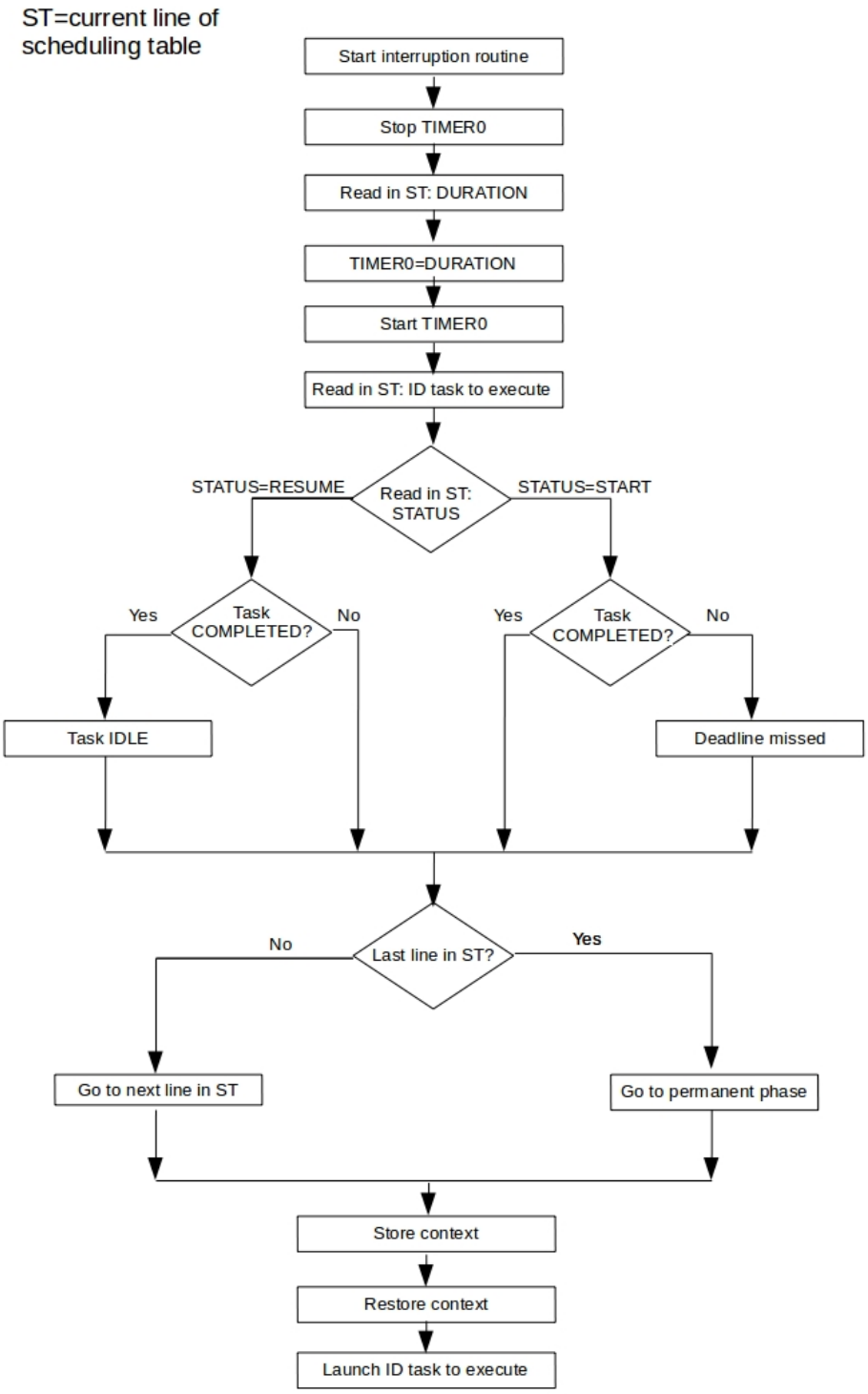


Figure 20: Flowchart of the interruption routine

We present also C and assembly codes of the scheduler.

```
//*************************************************************************
//* C and assembly code of the scheduler
//*************************************************************************

#include "headers.h"

#define EXCLUSIVE
#include "tableFinale.h"

const unsigned int COUNTER_VALUE[10] = {53,53,53,1,1,1,1,1,1,1}; // used by synthetic tasks

//***********************
Timer t;
uint16_t count=0;
int flag=0;
int j=0;
//***********************
Serial pc(USBTX,USBRX);
//***********************
DigitalOut SyncLED(LED1);
DigitalOut SyncLED2(LED3);

/***** Functions definitions  *****/

/***** assembly code *****/

extern "C"{
__asm void PendSV_Handler(void)
{ // Context switching code
// Simple version - assume No floating point support
// -------------------------
// Store current context
MRS R0, PSP ;// Get current process stack pointer value
STMDB R0!,{R4-R11}; // Save R4 to R11 in task stack (8 regs)
LDR R1,=__cpp(&scheduler.curr_task); // Get current task pointer address
LDR R2,[R1]; // Get current task pointer value
  STR R0,[R2, #__cpp(offsetof(task_t,psp))] //Save current PSP pointer value in current task psp field

// -------------------------
// Load next context
LDR R3,=__cpp(&scheduler.next_task); // Get next task pointer address
LDR R4,[R3]; // Get next task pointer value
STR R4,[R1]; // set curr task = next task
LDR R0,[R4, #__cpp(offsetof(task_t,psp))] //Load PSP pointer value from next task psp field
LDMIA R0!,{R4-R11}; // Load R4 to R11 from task stack (8 regs)
MSR PSP, R0 ;// Set PSP to next task
BX LR ;// Return
ALIGN 4

}
}

/***** Interrupt routine handler *****/

extern "C" {
 void TIMER0_IRQHandler(void)
{
LPC_TIM0->IR = 1UL << 0;
stop_TIMER0();

LOGTP(PREEMPTED);
LOGSCHEDB(BEGIN);

reset_TIMER0();
setPeriod_TIMER0(scheduler.sched_table[scheduler.index_table].duration);

start_TIMER0();
scheduler.next_task = scheduler.sched_table[scheduler.index_table].task ;
int resumed=0;
  if (scheduler.sched_table[scheduler.index_table].flag == 1)  { // if it is a new instance of the task
   /* verify if the task has completed its previous instance before beginning a new one */
if (scheduler.next_task->done == 0) // if not then call deadline missed task
```

```
{
scheduler.dead_id = scheduler.next_task->id;
scheduler.dead_id = 15;
scheduler.next_task = &dead_task;
}
else
{
miss = 0;
if ( scheduler.curr_task->id == scheduler.next_task->id)
{
scheduler.curr_task = &garbage_task;
}
}
STACK_FRAME_INIT(scheduler.next_task);

} else // if the status of the task is resumed, since it was preempted on its previous instance
{
miss = 0;
if (scheduler.next_task->done == 1) // the task is completed, then call the idle
{
scheduler.curr_task = &garbage_task;
scheduler.next_task = scheduler.task_tab[0];
STACK_FRAME_INIT(scheduler.next_task);
}
}

scheduler.index_table = (++(scheduler.index_table) == SIZE_SCHED_TABLE) ? I_PERM : scheduler.index_table ;

//if(index>=499) { //  loggin [size-1]
// exit(EXIT_SUCCESS);}
LOGSCHEDE(END);
if (resumed)
{
LOGTR(RESUME);
resumed=0;
}
SCB->ICSR |= SCB_ICSR_PENDSVSET_Msk; // Set PendSV to pending and call assembly code
}
}

/***** main function *****/

int main(void){

init_TIMER0();
init_TIMER1();

//INIT TASKS AND SCHEDULER
TASK0_CREATE();// idle
TASK1_CREATE();
TASK2_CREATE();
TASK3_CREATE();
TASK4_CREATE();
TASK5_CREATE();
TASK6_CREATE();
TASK7_CREATE();
TASK8_CREATE();
TASK9_CREATE();
TASK10_CREATE();

DEAD_CREATE();
SCHED_INIT();

//-------------------------------------------------------------------------
if (initLcdScreen()) {
GFX_INIT();
STACK_FRAME_INIT(scheduler.task_tab[0]);
STACK_FRAME_INIT(scheduler.task_tab[1]);
STACK_FRAME_INIT(scheduler.task_tab[2]);
STACK_FRAME_INIT(scheduler.task_tab[3]);
STACK_FRAME_INIT(scheduler.task_tab[4]);
STACK_FRAME_INIT(scheduler.task_tab[5]);
STACK_FRAME_INIT(scheduler.task_tab[6]);
```

```
__set_PSP(( scheduler.curr_task->psp + 16*4));  //__set_PSP((PSP_array[curr_task+1] + 16*4)); // Set PSP to top of task 0 stack
NVIC_SetPriority(PendSV_IRQn, 0xFF); // Set PendSV to lowest possible priority
__set_CONTROL(0x3); // Switch to use Process Stack, unprivileged state
__ISB(); // Execute ISB after changing CONTROL (architectural recommendation)


// setting and starting timer 0 & 1
LPC_TIM1->MR0 = 0xffffffff;
setPeriod_TIMER0(1);
start_TIMER0(); // timer for interruption
start_TIMER1(); // timer to keep track of time

while(1){} // the infinite loop

    }
else
{
printf("Couldn't start demo -> Initialization failed\n\r");
  }

   return 0;
}
```

References

[1] The scade tool page: http//www.esterel-technologies.com/products/scade-suite/.

[2] The simulink tool page: http://www.mathworks.com/products/simulink/.

[3] Theodore P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, 1991.

[4] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Trans. Softw. Eng.*, 21:475–480, May 1995.

[5] Giorgio C Buttazzo, Marko Bertogna, and Gang Yao. Limited preemptive scheduling for real-time systems. a survey. *Industrial Informatics, IEEE Transactions on*, 9(1):3–15, 2013.

[6] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time System.*, 2(3):181–194, sepember 1990.

[7] Maryline Chetto. *Real-time Systems Scheduling 1: Fundamentals*, volume 1. John Wiley & Sons, 2014.

[8] A. Choquet-Geniet and E. Grolleau. Minimal schedulability interval for real-time systems of periodic tasks with offsets. *Theor. Comput. Sci.*, 310(1-3):117–134, 2004.

[9] L. Cucu-Grosjean and J. Goossens. Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms. *Journal of Systems Architecture - Embedded Systems Design*, 57(5):561–569, 2011.

[10] J. Echague, I. Ripoll, and A. Crespo. Hard real-time preemptively scheduling with high context switch cost. In *Proceedings of 7th Euromicro workshop on Real-Time Systems*, Los Alamitos, CA, USA, 1995. IEEE Computer Society.

[11] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 301–310, Washington, DC, USA, 2010.

[12] J. Forget, E. Grolleau, C. Pagetti, and P. Richard. Dynamic Priority Scheduling of Periodic Tasks with Extended Precedences. In *IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA)*, Toulouse, France, September 2011.

[13] J. Goossens, E. Grolleau, and L. Cucu-Grosjean. Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. *Real-Time Systems*, 52(6):808–832, 2016.

[14] O. Kermia and Y. Sorel. A rapid heuristic for scheduling non-preemptive dependent periodic tasks onto multiprocessor. In *Proceedings of ISCA 20th International Conference on Parallel and Distributed Computing Systems, PDCS'07*, Las Vegas, Nevada, USA, September 2007.

[15] A. H. Kodancha. *Time Management in Partitioned Systems*. PhD thesis, Indian Institute of Science, Bangalore, Department of Computer Science and Automation, 2007.

[16] H. Kopetz. The time-triggered model of computation. In *Proceedings of the IEEE Real Time Systems Symposium, RTSS'98*, pages 168–177. IEEE Computer Society, 1998.

[17] J. Leung and M. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3), November 1980.

[18] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[19] F. Ndoye and Y. Sorel. Monoprocessor real-time scheduling of data dependent tasks with exact preemption cost for embedded systems. In *Proceedings of 16th IEEE International Conference on Computational Science and Engieering, ICCSE'13*, Sydney, Australia, December 2013.

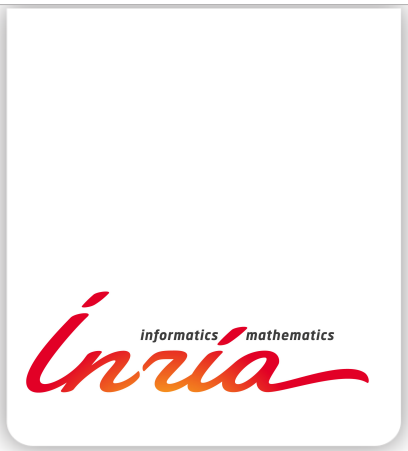
[20] J. M. Pont. *Patterns for Time-triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2001.

[21] P. Richard, Cottet, and M. Richard. On-line scheduling of real-time distributed computers with complex communication constraints. In *ICECCS*, pages 26–34, 2001.

[22] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39:1175–1185, 1990.

[23] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G.Bernat, C. Ferdinand, R.Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, G. Staschulat, and P. Stenström. The worst-case execution time problem: overview of methods and survey of tools. *Trans. on Embedded Computing Systems*, 7(3):1–53, 2008.

[24] P. Meumeu Yomsi and Y. Sorel. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In *Proceedings of 19th Euromicro Conference on Real-Time Systems, ECRTS'07*, Pisa, Italy, July 2007.



RESEARCH CENTRE
PARIS

2 rue Simone Iff - CS 42112
75589 Paris Cedex 12

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399